

## Topic 2

# Features of Software Development Process

---

### Contents

2.1	Introduction . . . . .	11
2.2	The need for iteration . . . . .	12
2.2.1	Review questions . . . . .	13
2.3	The Analysis Stage . . . . .	14
2.4	The Design Stage . . . . .	16
2.4.1	Review questions . . . . .	17
2.4.2	Designing the human-computer interface . . . . .	17
2.4.3	Designing Data Structures . . . . .	18
2.4.4	Other design choices . . . . .	19
2.5	The Implementation Stage . . . . .	20
2.5.1	Debugging . . . . .	21
2.5.2	Standard algorithms and module libraries . . . . .	21
2.5.3	Review questions . . . . .	23
2.6	Testing . . . . .	23
2.6.1	Test data preparation . . . . .	24
2.6.2	Alpha testing . . . . .	25
2.6.3	Beta testing . . . . .	25
2.6.4	Review questions . . . . .	26
2.7	The Documentation Stage . . . . .	26
2.7.1	User guide . . . . .	27
2.7.2	Technical guide . . . . .	28
2.8	Evaluation . . . . .	29
2.8.1	Robustness and reliability . . . . .	30
2.8.2	Review questions . . . . .	30
2.9	Maintenance . . . . .	31
2.10	Weaknesses of the software development process . . . . .	32
2.11	Summary . . . . .	33
2.12	End of topic test . . . . .	33

**Prerequisite knowledge**

*Before studying this topic you should be able to:*

- *describe the following stages of the software development process, analysis, design, implementation, testing, documentation, evaluation, maintenance;*
- *describe and be able to use test data (normal, extreme and exceptional);*
- *describe the features of a user guide and a technical guide;*
- *evaluate software in terms of fitness for purpose, user interface and readability.*

**Learning Objectives**

*After studying this topic, you should be able to:*

- *understand the iterative nature of the software development process*
- *describe each stage of the software development process*
- *explain the purpose of the software specification and its status as a legal contract*
- *understand and be able to describe corrective, adaptive and perfective maintenance*
- *explain the need for documentation at each stage of the software development process*

## Revision



**Q1:** The software development process consists of seven stages. Three of the stages are analysis, testing and maintenance. Which one of the following statements correctly identifies the missing stages, in order?

- a) Design, documentation, evaluation, implementation
- b) Design, evaluation, documentation, implementation
- c) Design, implementation, documentation, evaluation
- d) Design, documentation, implementation, evaluation

**Q2:** Which one of the following is an essential item of documentation that should be produced during the software development process?:

- a) Design notation
- b) Test data
- c) User interface
- d) User and Technical Guides

**Q3:** Which one of the following is NOT an aspect of the user interface of a program?:

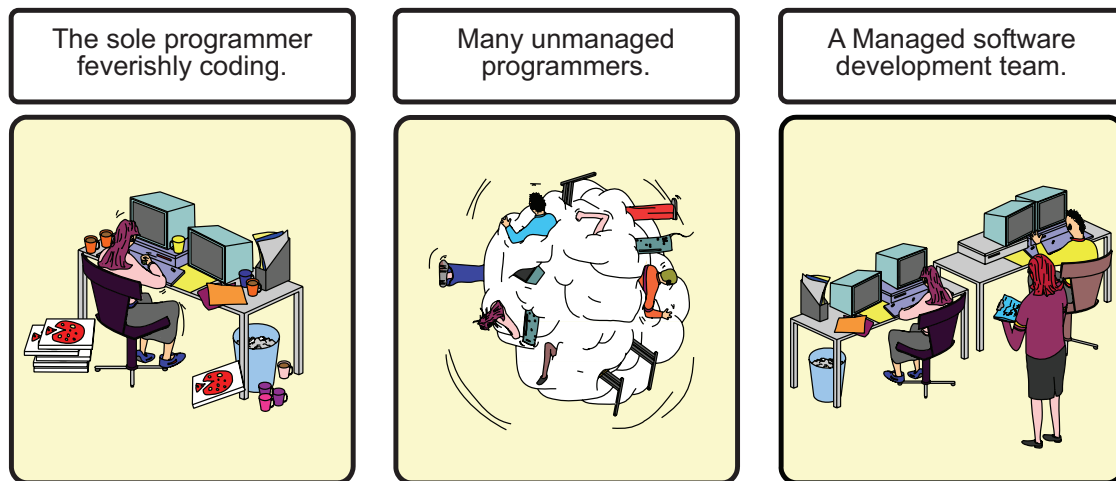
- a) Program listing
- b) Help screens
- c) Instruction screens
- d) Screen layout

## 2.1 Introduction

This topic considers the software development process. From the initial client specification to the production of a working program can take considerable time and effort by the development team. The process involves constant revision and evaluation at every phase which makes it an iterative process. This ensures quality and efficiency in the final product. Various models are introduced that aid the software development process but you will find that the perfect solution does not exist.

An individual may write a program for personal use. If it does not work then it can be changed. If more features or facilities are required then the individual can make amendments to their program.

This *ad hoc* method is not satisfactory in commercial environments where the goal is the creation of large scale software. A more structured approach is necessary.



Organisations creating software usually do so for profit. Money, time, and people are involved. The people involved have different points of view:

- some are clients, wanting to buy software;
- some are developers, concerned in creating software;
- managers are concerned with efficiency and profit within their organisation.

In the development of software, the three aspects which the developer must consider are:

- data
- processes
- human-computer interface

In traditional structured design, the primary tasks are to focus on the processes. A **process** is the work that a program carries out on data or in response to certain inputs.

The software development process does not always start in the same place. Sometimes there will only be an outline of the problem. At other times, a **specification** will be available.

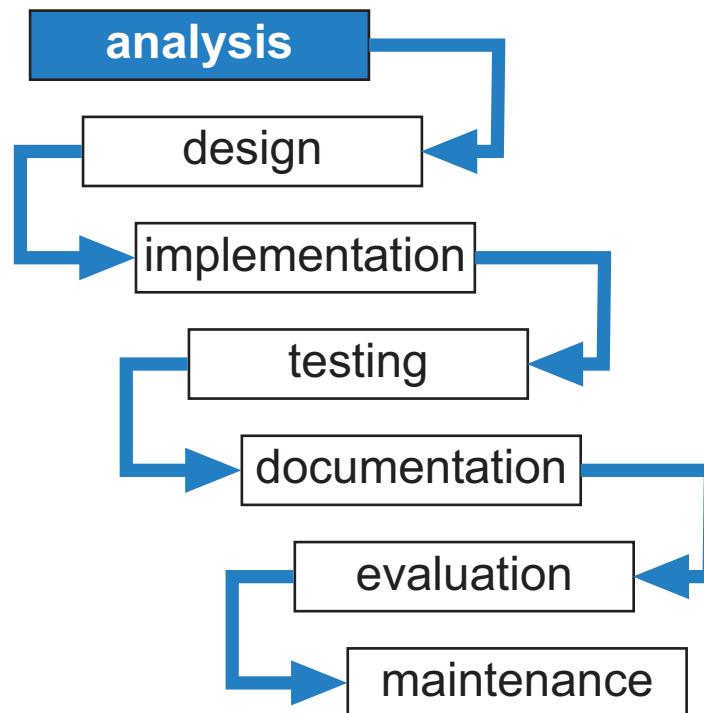
The specification must be agreed with the clients. Work on the problem and the solution is often carried out by a group of people, called the **development team**.

The aim of the team is to produce a new software system that will solve the problem.

## 2.2 The need for iteration

The traditional software development process contains a sequence of stages. The precise names of the stages, and even the number of stages, are not universally agreed. They differ from book to book and from developer to developer. Much depends on the aspects of software development that a book wishes to emphasise or that a particular developer prefers.

In this course we will consider the following stages:



The idea here is that the development of the system flows down or cascades through the stages like water flowing down a fall. As each stage is completed responsibility and control is passed down until the final section is completed.

The process, as it stands, represents an ideal world rather than reality. Developers do not know everything the client will need at the start of a project; they make wrong decisions, possibly based on incomplete information.

The model can be improved significantly if it is made to be **iterative**.

Ideally, you would start a process with the analysis and work through the stages in turn, doing everything only once. In practice, this happens rarely. People make mistakes, faults become apparent that can only be corrected by going back to an earlier stage of the process.

### An iterative development model

There is an on-line illustration of the need to use an iterative development model. You should view this animation now.



#### 2.2.1 Review questions

**Q4:** Which one of the following statements regarding the order of the stages in the software development process is correct?

- a) Design, implementation, testing, documentation
- b) Analysis, testing, implementation, design
- c) Design, testing, evaluation, implementation
- d) Analysis, design, testing, evaluation

**Q5:** The software development process may involve iteration. Which one of the following statements is true if iteration is used?

- a) The team may go back to an earlier stage to deal with a problem.
- b) The software is guaranteed to be error free.
- c) Each stage is carried out twice.
- d) Once all the stages have been completed, the team go back and start again.

**Q6:** Software developers cannot get the software correct at the first attempt. This is because:

- a) Software systems can be very complex
- b) Unforeseen errors always creep in that take time to solve
- c) There could be problems with the client changing his/her mind
- d) All of the above

**Q7:** Regular feedback of information to members of the development team is important. This is done in order to:

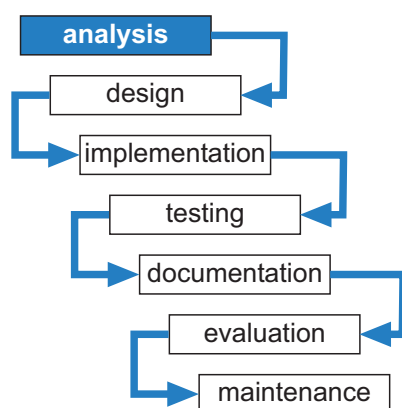
- a) Speed up the development process
- b) Enable personnel to discuss progress
- c) Keep the team happy
- d) Lower the costs on a regular basis



### Sentence completion - software development process

On the Web is a sentence completion task on the software development process. You should now complete this task.

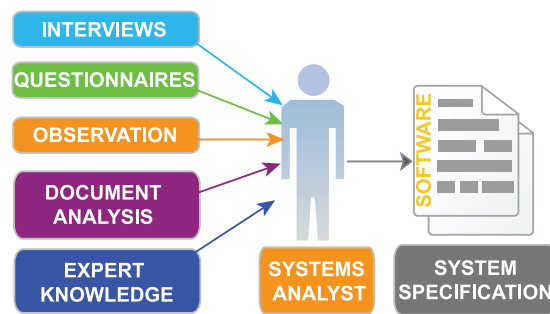
## 2.3 The Analysis Stage



This stage is crucial to the entire cycle of events. Any problems occurring at this stage will be propagated through the system and will become increasingly costly to rectify when discovered.

Analysis is an attempt to understand a given problem, clearly and exactly, in order to generate a solution. The outcome will be a specification that is used as the basis for all subsequent work.

Analysis, sometimes called systems analysis, is the job of a specialist person - the systems analyst.

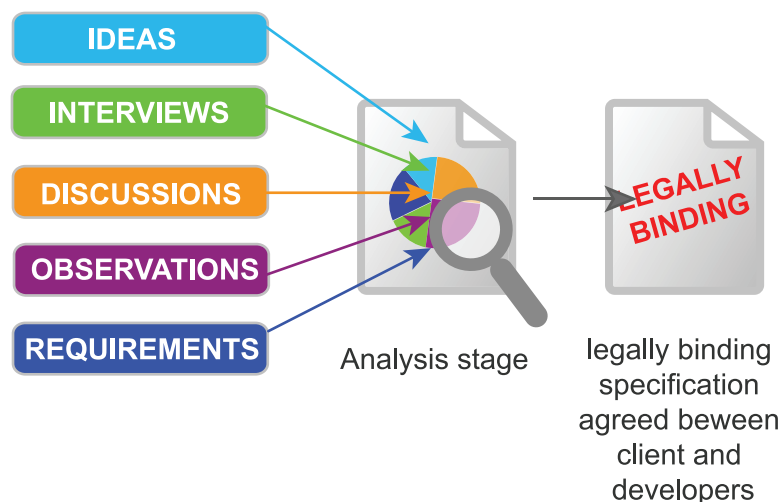


Full systems analysis has three phases:

1. collection of information
2. analysis of information collected
3. production of a problem specification or user requirements specification.

Sometimes, this stage begins with a vague idea or rough outline of the problem. On other, rather more formal, occasions, it will start with a full **requirements specification**. This will include:

- hardware and software specifications,
- notes on project issues such as,
  - objectives,
  - constraints,
  - costs
  - and schedule.
- It may also include a full **functional specification**, which will describe exactly how the system is meant to behave. The functional specification is what the development team will follow in creating the software system.

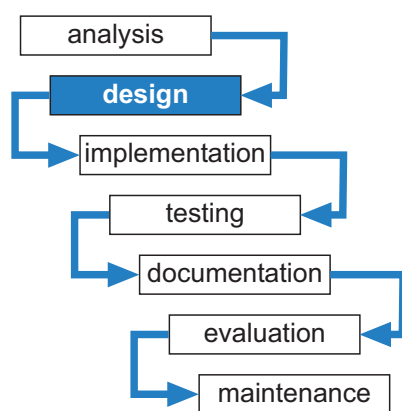


Questions to be asked at this stage would include:

- What are the new system requirements?
- What are the costs involved?
- How long will it take to implement?

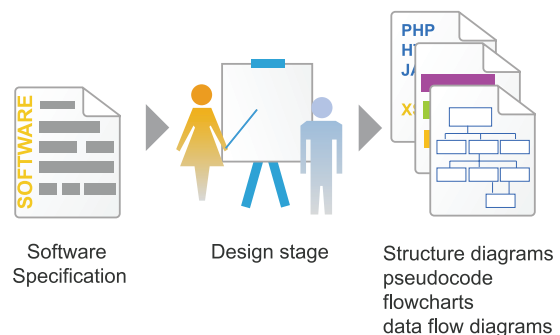
Details are gathered by a variety of methods such as interviews and questionnaires.

## 2.4 The Design Stage



The product of the analysis stage is the completed specification. During the design stage, the specification of the problem is used as the basis for a planned solution.

The work on the design is carried out by a designer: this might be same person as did the analysis or it may be a member of the programming team. Most analysts can program and many programmers can carry out analysis. People who can do both are described as analyst/programmers.



The design process is methodical, using techniques such as structure charts and pseudo-code. The problem is approached by breaking it down into a collection of relatively small and simple tasks. This approach is known as top-down design with stepwise refinement.

The development team will attempt to make the design of the program both robust and reliable.

- A reliable program is one that does not stop because of faults in its design
- A robust program is one that can cope with errors when it is running

To put it another way, an unreliable program is one that hangs or crashes for no apparent reason whereas a non-robust program is one that cannot cope with events that the world

throws at it.

### Identifying the characteristics of good software design



On the Web is an interactivity. You should now complete this task.

#### 2.4.1 Review questions

**Q8:** Which one of the following processes describes breaking a complex system down into more manageable components?

- a) top-down design
- b) using pseudocode
- c) refined design
- d) prototyping

**Q9:** In the software development process which of the following is a legal contract?

- a) functional specification
- b) problem specification
- c) requirements specification
- d) software specification

**Q10:** Which of the following are identified during the analysis stage of software development?

- a) the main costs of the project
- b) time taken to complete the project
- c) hardware required to run the system
- d) All of the above

**Q11:** The completed software system should be able to cope with many errors while running. This means that the software is (choose one):

- a) Portable
- b) Robust
- c) Reliable
- d) Stable

**Q12:** Which of the following is included in the **functional specification**?

- a) a description of what the software must do
- b) the hardware used to run the software
- c) the nature of the problem to be solved
- d) an outline of the problem solution

#### 2.4.2 Designing the human-computer interface

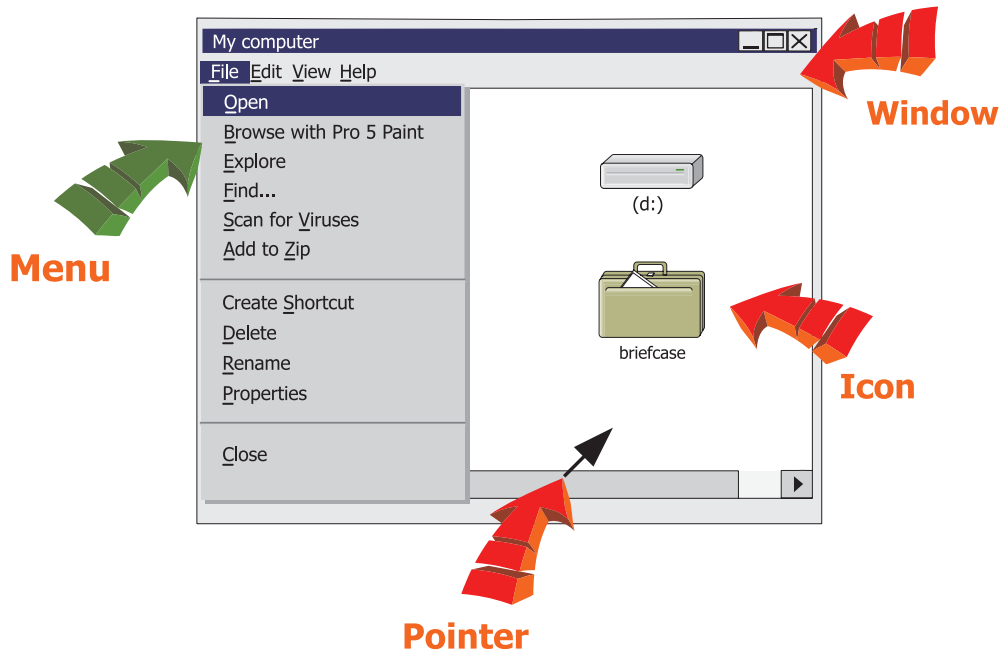
The **human computer interface** is all that a user sees of a program.

A program's viability often relies on the quality of the HCI. A good interface makes things easier for the user. A bad interface can introduce mistakes and cause irritation.

Modern HCIs are often designed as Graphical User Interfaces (GUIs) which provide a WIMP environment. WIMP stands for Windows Icons Menus Pointers (although some

textbooks refer to Windows, Icons, Mouse, Pulldown menus). A GUI provides a set of Windows, which contain Icons and Menus. The user controls the program by means of a Pointer which reflects movement of a mouse, trackerball or other input device.

## The WIMP environment



The HCI must allow easy navigation. Users should be able to move from one screen to another in a straightforward manner and to leave screens when they wish. Some sites on the Internet, for example, include a site-map to show the user how different forms are linked.

HCI must be consistent, so that similar actions in different parts of the interface have similar responses throughout the program. Prompts given to the user should be consistent. Different screens should look as though they belong to the same software package.

The HCI should provide on-line help, offering intelligible prompts and send messages and warnings to the user about the consequences of choices made, *e.g. send a warning if a user chooses to delete data.*

HCI design is based on an appreciation of what the user wishes to see. The designer thinks in terms of the windows (often called forms) that are presented to the user.



### Identifying characteristics of a good user interface

On the Web is a interactivity. You should now complete this task.

#### 2.4.3 Designing Data Structures

A program must perform operations on the data supplied to it. The data should be structured data. The choice of data structure will affect the entire program.

When the amount of data is likely to be very large, the designer must consider the

physical capacity of the hardware. For example, a million records, of a thousand bytes each, will require about a gigabyte. If the clients want these records ordered in different ways in different parts of the program, even a modern PC may have insufficient memory.

Many large systems involve a **database**. In many cases, the data is held in different tables which are linked together. These links are called relations and such a database is known as a relational database. The fields that are to be in the tables and the relations between tables need to be defined at the design stage.

Object oriented design attempts to treat data and **objects** together. An object brings together items of data and the operations that can be carried out on it. For example, a data item might be a customer's record, and the operations might include creating, displaying, editing, and deleting.

#### 2.4.4 Other design choices

The system specification and functional specification will form the basis for designing the program. The design team must consider:

- hardware specification
- choice of high level language
- choice of operating system
- portability of the system

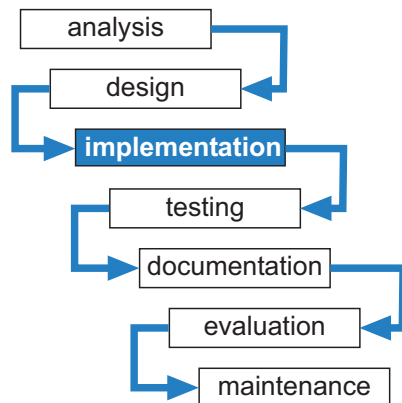
Hardware aspects will include processor speed (does the program require multiple processors for optimum performance as in networked database applications or maybe maximum memory for caching information on a regular basis). How much external storage is required for, say, regular backing up procedures and on what medium? This is an important issue especially on networked systems where users' files are archived on a daily basis.

A high level language will be chosen that is best suited to the problem but also one which the programming team is conversant with and proficient in its use. Modularity will be an important issue where the team can share the workload by compiling modules independently thereby reducing the overall development time. Module libraries can be a source of standard algorithms to be used in software projects of many types.

Choice of operating system will relate to the functionality and feel of the HCI. Windows might offer much in the way of colourful screens, interactive help and dialogue boxes etc. It must also affect how the program runs and behaves: is the OS software stable enough for the developed program to behave normally without crashing or does the OS offer a true multi-tasking environment, as in UNIX with the added benefits of in-built security. Nowadays Linux is being seen as a viable operating system which is both stable and not difficult to use.

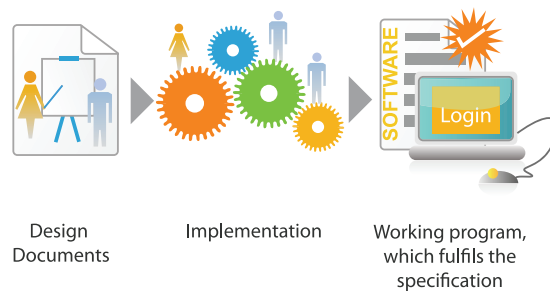
If the developed software can be moved to different machine architecture and still run to specification then it will be deemed to be portable. In some cases this might not be required but it is a characteristic that good software should possess.

## 2.5 The Implementation Stage



If the design has been thorough the implementation should be straightforward. It should be a matter of translating the pseudo-code into code, line by line.

The design is implemented when it has been converted into code which can be used by the computer system.



The code is written in a **high-level language**, such as Pascal, C++ or Java, and converted into code which the computer understands.

A high-level language is one that people find relatively easy to understand. The code written at this stage is called **source code**.

The machine can understand **machine code**, a translation of the source code into binary instructions. This code, because it can be executed by a computer, is also known as **executable code**.

The translation, from source code into machine code, is carried out by a program called a **compiler**. The resultant machine code is **portable** to machines of the same type running under the same kind of operating system.

Compiling and debugging large programs can take a lot of time.

Another form of translation that can reduce development time is an **interpreter**. When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter is also executing each line of code one at a time the programmer is able to see the results of the program immediately which can also help with debugging.

Sometimes problems that arise at the implementation stage will call for a return to an earlier stage of the development process. For example, it might turn out that an

algorithm runs too slowly to be useful and that the designer will have to develop a faster algorithm. Or it might be that the slowness of operation is due not to a poor algorithm but to the hardware capability. In such a case, the development team might have to return to the analysis stage and reconsider the hardware specification.

At the end of the implementation stage a structured listing is produced, complete with internal documentation. This will be checked against the design and against the original specification, to ensure that the project remains on target.

### 2.5.1 Debugging

At this stage, the **programming team** will make use of **test data**.

This data is designed to check that the program works properly, and that it is reliable and robust. **Testing** is often confused with the **debugging** of a program, but these are not the same, though they are very closely related.

- testing discovers any faults in a computer program
- debugging is the finding and correcting of these faults.

Maintainable software should include **internal documentation**. This is commentary within the program to explain the various stages and to record any changes that might be implemented in the coding during debugging. One aspect of this is the use of meaningful variable names.

### 2.5.2 Standard algorithms and module libraries

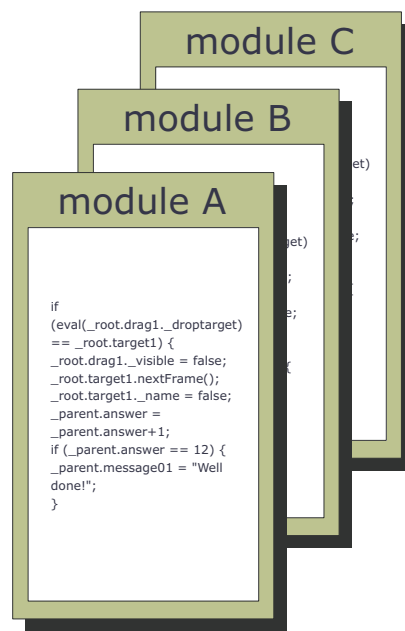
#### Standard algorithms

Most projects will use certain standard algorithms. Programmers need to be familiar with these common algorithms. Ones that you will become familiar with later include:

- linear searching
- counting occurrences
- finding maxima and minima

#### Module libraries

It is often possible to use, with or without alteration, modules that have been previously written and have been retained in a module library. A module library will include code for standard algorithms. Most development environments come with a large library of modules. Programmers can use these in the code they are developing. These libraries will include mathematical functions, modules for converting text to numbers, etc.



Each module in a module library is:

- pre-tested
- well documented

so that it can be adapted and easily used.

The use of previously written modules reduces the time spent creating the final software and minimises the cost. Designers will incorporate, if possible, modules from the module library in the design.

For many software companies and programming departments, one project will be similar to another. Projects will have component parts which are similar; for example many programs might require a sort routine to act on the contents of a database.

As time goes by, a collection of modules is built up. This collection is known as a **module library**.

Modules in the library can often be used, over and over, in new projects. This saves time spent designing and coding. If a module can be used without alteration, there is the added advantage that it has been thoroughly tested and is reliable. Use of unaltered modules reduces the time spent on the detection and correction of errors in the project as a whole.

Sometimes adjustments might be needed to a library module. After the necessary work, the module will require to undergo error and other testing.



### Characteristics of module libraries

On the Web is a interactivity. You should now complete this task.

### 2.5.3 Review questions

**Q13:** Which of the following is a quality of a good human computer interface?

- a) It should be Windows-based
- b) It can make a program execute faster
- c) It can make running a program a less irritable experience
- d) It should have lots of colour

**Q14:** Which one of the following is not a high level language?

- a) Visual Basic
- b) Pascal
- c) Assembler
- d) C++

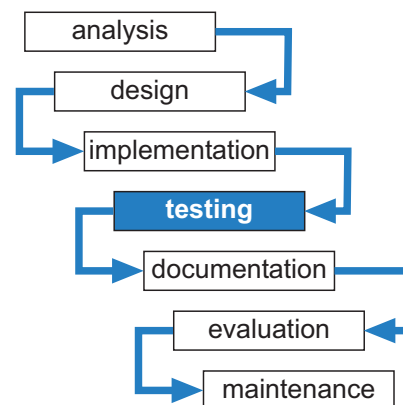
**Q15:** Which of these is not the result of compilation?

- a) Executable code
- b) Object code
- c) Machine code
- d) Source code

## 2.6 Testing

Testing has several purposes. It should check that:

- the software meets the specification
- it is robust
- it is reliable.

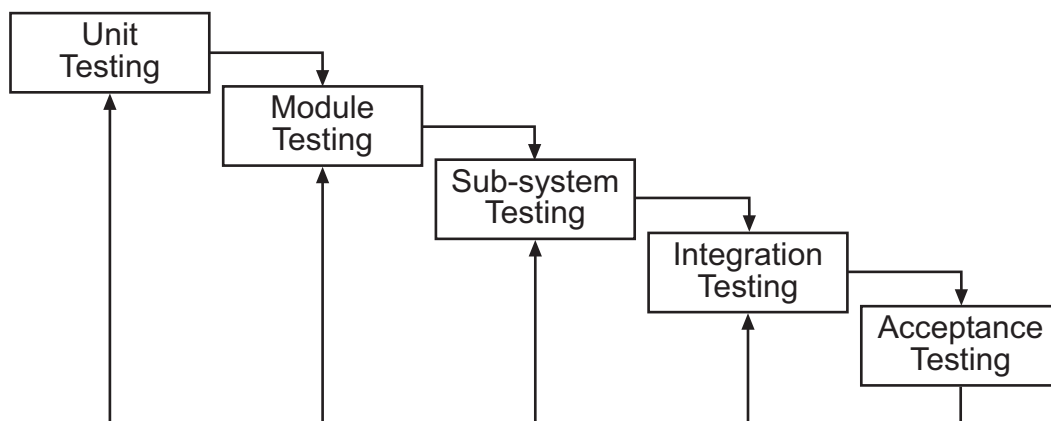


Commercial software is *not* exhaustively tested at the testing stage. Software can be complicated and the available time is limited. There must be a balance between creating a product for the market and **exhaustive testing**. If errors become apparent after release, the company will fix them and release an updated version.

Testing follows a **test plan** or strategy, involving carefully selected test data, with a view to ensuring that a reliable product has been constructed. Important aspects would be:

- what part of the program is being tested?
- what is the expected output using suitable test data?

Testing is carried out at several stages during and after implementation:



### 2.6.1 Test data preparation

Testing can never show that a program is correct. Even with extensive testing, it is almost certain that undetected errors exist. Testing can only demonstrate the presence of errors, it cannot demonstrate their absence.

Tests should be devised against the specification, so that you can see whether or not the program does what it is supposed to do.

In the full software development process, test data should be prepared before implementation. That is, before you have invested time and effort in writing the code.

All too often there is a temptation especially at classroom level to start coding a program as soon as possible without having produced adequate test data.

But experience shows that this is a mistake. Once you have written the code you will tend to go easy on it, and let the program's behaviour shape what you expect of it. You are going to be too kind to it.

MURPHY'S LAW 1	MURPHY'S LAW 2
The quicker program coding is started the longer the project will take	Murphy's Law 1 is correct!

Testing follows a test plan or strategy. Testing should be systematic. That means:

- testing is planned,
- everything is tested in a logical order,
- the results of testing are recorded.

With most software projects, the usual strategy is to test the software twice. The methods are called:

- **alpha testing** where the software is tested *within* the organisation
- **beta testing** where the software is tested by personnel *outside* the organisation or by certain members of the public. This is sometimes called **acceptance testing**.

### 2.6.2 Alpha testing

Test data is based on the specification. Data will be designed to test three aspects of the program:

- **normal operation:** data that the program has essentially been built to process; all outputs should be satisfactory.
- **extreme / boundary testing:** data to test that the program functions properly with data at the extremes of its operation; for example, if a number entered is meant to be limited, the program's performance is tested just within the limit, on the limit, and just beyond the limit; as another example, if a table is supposed to have a maximum number of elements, the program is tested to see if it can cope with exactly the maximum and if it can cope when an attempt is made to exceed the maximum.
- **exceptions testing:** data that lie beyond the extremes of the program's normal operation; these should include a selection of what might be called silly data, to test the program's robustness, that a user might enter in a moment of confusion or mischief.

### Matching definitions - Testing

On the Web is a interactivity. You should now complete this task.



Faults that become evident during testing are known as **bugs**. If bugs are identified, the program is sent back, with the test logs, to the programming team for **debugging**. This process is likely to be iterative: testing, finds bugs, they get fixed, the program's tested again, more bugs are found, and so on.

It may be that bugs reveal flaws that were introduced at an earlier stage of the process, at the design or even at the analysis stage. If this is the case, the documentation for each stage of the development process will need to be corrected.

### 2.6.3 Beta testing

Otherwise known as acceptance testing it takes place after alpha testing. The idea is to subject a completed program to testing under actual working conditions.

If a program has been developed for use by particular clients, it is installed on their site. The clients use the program for a given period and then report back to the development team. The process might be iterative, with the development team making adjustments to the software. When the clients regard the program's operation as acceptable, the testing stage is complete.

If a program is being developed by a software house for sale to a market rather than an individual client, the developers will provide an alpha-tested version to a select group of expert users such as computer journalists and authors, and also makers of related computing products such as printers. This group is known as an independent test group. Alternatively, they may use professional software testers.

This is of benefit to both parties: the software house gets its product tested by people who are good at noticing faults, and the writers get to know about products in advance; which further benefits both parties when the final production software is released, the

software house getting publicity and the writers receive credit for being up to date.

People involved in beta testing will send back error reports to the development team. An error report is about a malfunction of the program and should contain details of the circumstances which lead to in the malfunction. These error reports are used by the development team to find and correct the problem.

#### 2.6.4 Review questions

**Q16:** Which one of the following statements describe alpha testing?

- a) Testing is done by the users
- b) Testing is done within the organisation
- c) Testing is done by specialist companies
- d) Testing is done by the client

**Q17:** Which of the following describes beta testing?

- a) The program is tested by the clients
- b) The testing is more rigorous than alpha
- c) The testing is for market research
- d) The program is tested by specialist companies

**Q18:** During alpha testing, the program is usually subjected to **exceptions testing**. This means:

- a) The input of unexpected data
- b) The input of large numbers
- c) The input of small numbers
- d) All of these

## 2.7 The Documentation Stage

Documentation is intended to describe a system and make it more easily understood. Documentation will consist of:

1. user guide
2. technical guide

Some information may appear in both guides; e.g. system specification.

**Internal documentation** such as remarks or comments in the code are for the benefit of the development team. It will help if changes have to be made to the software in the future.

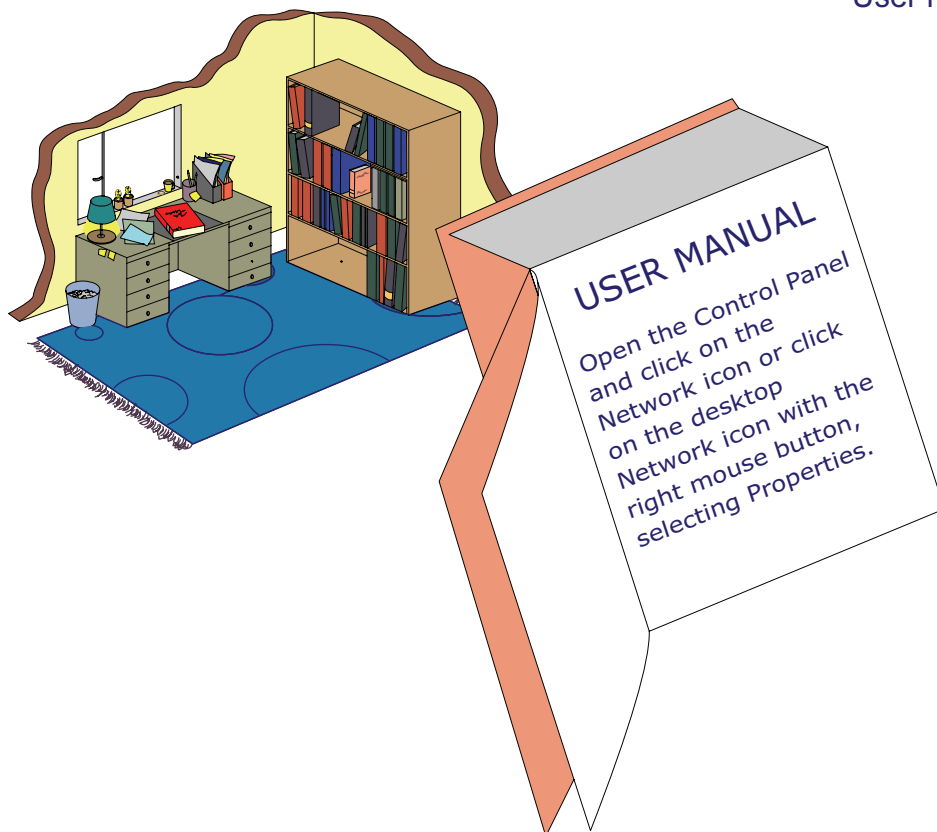
Other documentation for the development team includes all the documents produced in the software development process: requirements specification, program design documents (for the HCI and for the structure and logic of the underlying code), a structured listing of the code, and a test history.

### 2.7.1 User guide

The user guide contains information about how to install, start and use software. It should also contain a list of commands and how to use them. Where there is a significant HCI, the guide will show each form, menu, and icon, and associated instructions about their use.

User guides may be supplied as paper manuals, often with separate manuals for installation, getting started and the user guide.

User manual



Increasingly software vendors supply the manuals on disc or CD where they may be available in (pdf) or hypertext markup language (HTML).

Paper manuals are costly to reproduce; manufacturers frequently include electronic files which provide up-to-date amendments. This lets the user read up to date information which could not be included in the original paper manual.

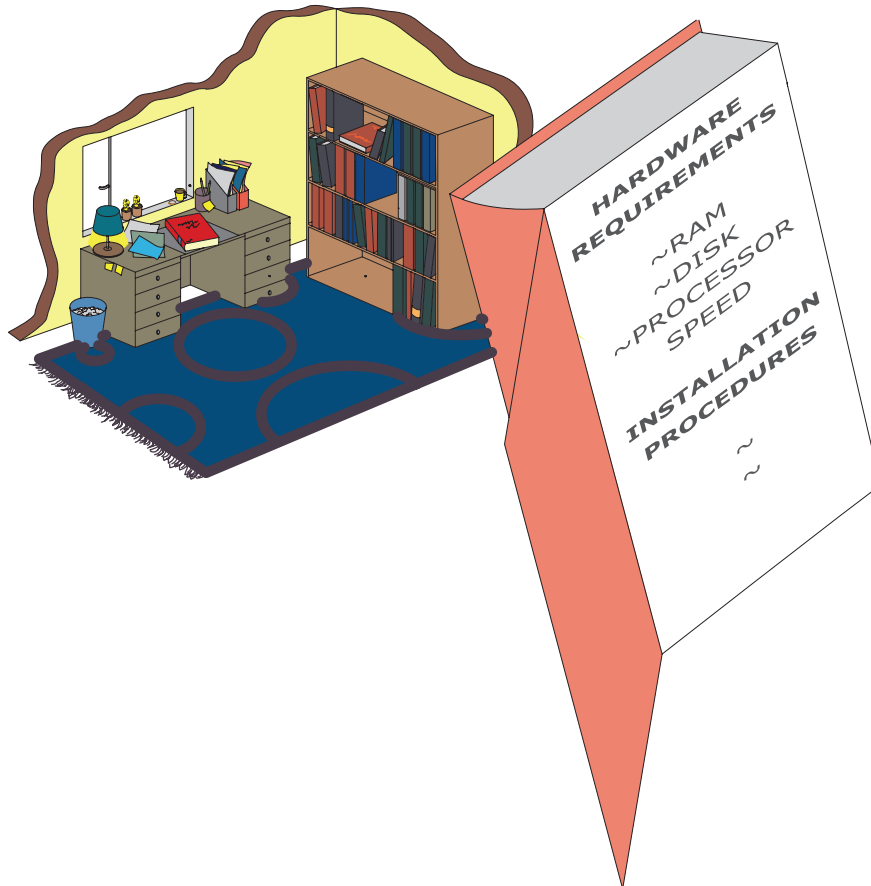
It is common for sample files to be included which complement the tutorial and provide the user with demonstration files.

The program should contain a help facility. It is common for on-line help to be presented in three tagged pages: Contents, Index, and Search. The contents present the help chapter by chapter; the index refers to certain key words in the chapters; and search offers the facility to locate key terms within the guide.

### 2.7.2 Technical guide

The technical guide will contain information about the hardware and software requirements of the program. The hardware specification will include details of the processor type and speed, RAM required, RAM desired, monitor resolution, graphics and sound card specifications etc. It will also contain instructions about configuring the program.

#### DOCUMENTATION



Software designed to operate, or run on networks can be very complicated and require a good deal of expertise. Technical guides can be very large and cumbersome and difficult to navigate.



#### Sentence completion - Documentation

On the Web is an interactivity. You should now complete this task.

## 2.8 Evaluation

Evaluation is the formal monitoring of a system to ensure that it is performing its purpose accurately, efficiently, cost effectively and in a timely manner. The performance of the system must be matched against a given set of *criteria* such as the initial project specification.

Evaluations of various kinds are an important aspect of the software development industry. Evaluations are used to determine if systems are usable, cost effective, conforming to performance criteria, etc. Evaluation is based on observation, interviews, and questionnaires. Additionally techniques such as automatic data logging are used. Many organisations bring in consultants who design and carry out evaluations as the skills required to carry out effective evaluations are highly specialised.

The key criterion in evaluating a software product has to be whether it is **fit for purpose** i.e. does it meet the original specification and allow the client to carry out their tasks?

The evaluation is important for the user and the software author. There are two reasons for conducting an evaluation:

- does the software meet the users' requirements?
- how can the software house improve the product?

The performance of the system can be assessed in various ways:

- how closely does it fit the system design?
- how well does it meet the problem specification?

Questions may also be asked about matters such as:

- was the project within budget?
- was the project completed to schedule?

The development team will wish to review the project, perhaps to learn from any mistakes to ensure they incorporate good points in future programmes.

Software houses aspire to produce new and better versions of their software. They will study press reviews and note any contents and criticisms. New or forthcoming changes (in technology, in operating environment, and so on) are also taken into account. When the evaluation is complete, work begins on the next version of the system.

For SQA assessment purposes, you need to be able to evaluate software in terms of:

- robustness
- reliability
- portability
- efficiency

- maintainability  
(and from Int2 level)
- readability
- fitness for purpose, and
- quality of HCI



### Sentence completion - Evaluation

On the Web is an interactivity. You should now complete this task.



### Evaluation terminology

On the Web is an interactivity. You should now complete this task.

## 2.8.1 Robustness and reliability

There is often confusion between the terms robustness and reliability.

A program is robust if it can cope with problems that come from outside and are not of its own making e.g. *corrupt input data*. Reliability is an internal matter. A program is reliable if it runs well, and is never brought to a halt by a design flaw.

When the program is complicated the distinction between the two terms is not always clear. When a machine hangs it is not always obvious whether this is due to a failure in robustness or reliability.

### Robustness

The designer should try to ensure that the design is **robust**: the resulting software should be able to cope with mistakes that users might make or unexpected conditions that might occur. These should not lead to wrong results or cause the program to hang. As examples of an unexpected condition, we could take something going wrong with a printer (it jams, or it runs out of paper) or a disc drive not being available for writing, because it simply isn't there (the user's forgotten to put in the CD), or the user entering a number when asked for a letter.

### Reliability

A **reliable** program always produces the expected result when given the expected input. It is designed correctly to do the task specified.



### Characteristic of good software design

On the Web is an interactivity. You should now complete this task.

## 2.8.2 Review questions

**Q19:** Choose the correct response that describes the term **robust** within software development:

- The program is strong and hardy
- The program may be ported to a different machine architecture
- The program can cope with mistakes that the user might make

d) The program runs to specification

**Q20:** Choose the correct response that describes the term **reliable** within software development:

- a) The program is strong and hardy
- b) The program may be ported to a different machine architecture
- c) The program can cope with mistakes that the user might make
- d) The program runs to specification

**Q21:** Using a module library can reduce software development time because (choose one):

- a) Common programming modules can be used
- b) Programmers do not need to write programs from scratch
- c) Programmers can re-use library code in their projects
- d) All of the above

## 2.9 Maintenance

Once the software is operating, the users will need support. In the case of a bespoke system, the development team (or the organisation it works for) may offer training in the use of the new system.

Creators of software systems often establish help desks, so users can obtain advice about the software.

Software does not wear out, in any physical sense, but the presence of errors or omissions will give rise to the need for **maintenance**.



Software maintenance always involves a change in the software with the accompanying probability that additional errors may be introduced. It is essential to ensure that adequate quality control is in place.

There are three types of software maintenance:

- corrective
- adaptive
- perfective

**Corrective maintenance** is concerned with errors that escaped detection during testing but which occur during actual use of the program. Users are encouraged to complete an error report, stating the inputs that seemed to provoke the problem and any messages that the program might have displayed. These error reports are invaluable to the development team, who will attempt to replicate the errors and provide a improved solution.

**Adaptive maintenance** is necessary when the program's environment changes. It allows the authors to provide a program which responds to changes in the operating environment. For example, a change of operating system could require changes in the program, or a new printer might call for a new printer driver to be added to the program. A change of computer system will require the program to be ported to the new system.

**Perfective maintenance** occurs in response to requests from the user to enhance the performance of the program. This may be due to changes in the requirements or new legislation. Such maintenance can involve revision of the entire system and can be expensive.



### Matching definitions - Maintenance

On the Web is an interactivity. You should now complete this task.

## 2.10 Weaknesses of the software development process

Problems are usually encountered when you use this model:

1. Real projects rarely follow a linear, sequential flow. Apart from any software problems, people change their minds, and often there may be changes in legislation which mean that the program must be altered in order to comply with the new regulations. No matter what the reason, iteration always occurs and this creates problems because much of your work has to be re-examined and revised;
2. It is difficult for the customer to state all requirements explicitly at the start of developments. The model depends on this and has difficulty incorporating customer uncertainty;
3. Clients are frequently excluded from the development. The working version of the program will not be available for the customer to see until late in the development cycle;

4. Developers work in isolation from the clients, often for months, only for the clients to be disappointed with the results. Many developers value feedback from clients as the project progresses;
5. Errors arising from incorrect requirements will not be obvious until late in the cycle, by which time they will be difficult and expensive to fix. There is nothing so depressing as delivering months, sometimes years, of work to the customer only to be greeted with the response, "That's not what I wanted at all."

## **2.11 Summary**

The following summary points are related to the learning objectives in the topic introduction:

- the software development process has 7 stages (analysis, design, implementation, testing, documentation, evaluation, maintenance);
- the process is iterative and involves a continual revision and evaluation at each phase of the process.

## **2.12 End of topic test**

An online assessment is provided to help you review this topic.

